Allister Liu, Amy Leong, Steven Lee

ECE-357 Computer Operating System

Problem Set #4 Part 1&2

1a) What key sequence would you use to cause a foreground process to terminate and dump core (if possible)?

- Control + C, this key sequence sends SIGINT which will terminate the process.

1b) When would a signal SIGTTOU be sent to a process? What happens to the process when it receives it (assuming that this signal's disposition is default)?

- SIGTTOU is sent to a process when a process in a background job attempts to write to the terminal or set its modes. It is the signal that tells the process to stop waiting for tty output. By default, it will stop the process.

1c) Process "A" sends (using the kill system call) signal #40 to process "B". It does so 3 times in a row. At the time, process "B" has signal #40 in the blocked signals mask and has set up a handler for this signal. At some later time, process "B" unblocks signal #40. What happens and why?

- Process "A" sends signal #40 to process "B" for three times in a row. All three signals are queued into a sigqueue because they are real time signals. Due to them being blocked, they will not be forgotten, but will stay until the process is unblocked, this is because the corresponding bit in the pending signals array is set first. In a later time when the signals are unblocked, the three signal #40s are executed one by one (switching to kernel mode to execute the first signal #40 then back to user mode; switching to kernel mode to

execute the second signal #40 then back to user mode again; switching to kernel mode to execute the last signal #40 and finally back to user mode again.)

1d) What does it do? What is its wait exit status? Don't just tell me, explain what is happening and why.

- The program first set a char pointer p to null pointer, this means that both *p++ and *p-- operations are illegal, causing an SIGSEGV signal. By using signal(), the program then sets up the handler function whenever an SIGSEGV is received. In the condition of the "if" statement, the long jump point is set (returning 0 so we do not go into the "if" statement). Proceeding to the line *p--, an SIGSEGV signal is generated, and we are redirected to the handler function, and the SIGSEGV signal is being blocked upon entry. Since there is a longjmp in the handler function, the restoration of the blocked signals mask does not happen and SIGSEGV would be set. In the handler function, "In handler instance 1" is printed to standard error. Then we long jump to the setjmp in the if statement, and prints "The other side" to standard error. After that, the program encounters a segmentation fault error when executing *p++, and exits with an exit status of 139 (128+11). 128 is from the core dump file being generated and the 11 is from signal #11. Since signal() is used the behavior is undefined when the eSIGSEGV signal was not generated by kill(2) or raise(3), thus leading to a segmentation fault (core dump) error message.

2a) Can this program ever produce as part of its output the sequence ABA? Explain why or why not.

- It can not produce the ABA, but it can have a possibility of 1024 As then 1024 Bs then 1024 As. Since the child processes are running in no particular order, there is a possibility that it can be overlapping.

2b) What happens if I remove the line of code marked THIS LINE? Explain why

- When the close(pp[1]) line is removed, the write side of the pipe is not closed. The program will still believe that there will be input coming in and thus it will be hanging forever because no EOF is detected and the while loop will not end.